Lecture 19 — Homomorphic Encryption 1: Definition and Application to Private Information Retrieval, Zero knowledge.

Boaz Barak

April 12, 2010

Homomorphic encryption We stressed that CPA security does not prevent an attacker from tampering with the encrypted message, changing for example an encryption of the message x into an encryption of x with its last bit flipped. Homomorphic encryption takes this to an extreme and actually *requires* that it is possible to tamper with the encryption in an arbitrary way (while still maintaining CPA security!). The question if this is possible was first raised in 1978 by Rivest, Adleman, and Dertouzos, and over the years many conjectured that this is in fact impossible. Last year Gentry gave very strong evidence that such encryptions exist, by constructing such a scheme that is secure under relatively reasonable computational assumptions.

Definition We say that a CPA-secure public key encryption scheme (G, E, D) with one bit messages is *fully homomorphic* if there exists an algorithm $NAND$ such that for every $(e, d) \leftarrow G(1^n)$, $a, b \in \{0,1\}$, and $\hat{a} \leftarrow E_e(a)$, $\hat{b} \leftarrow E_e(b)$,

$$NAND_e(\hat{a}, \hat{b}) \approx E_e(aNANDb)$$

where $\approx$ denotes statistical indistinguishability (i.e., $n^{-\omega(1)}$ statistical distance), and $aNANDb$ denotes $\neg(a \wedge b)$.

(There are several variants of the definition, and we chose the simplest and strongest one.)

We stress that the algorithm NAND does *not* get the secret key as input. Otherwise it would be trivial: just decrypt $\hat{a}, \hat{b}$, compute $aNANDb$ and re-encrypt.

Below we'll often drop the adjective "fully".

Universality of NAND It's straightforward to show that every log gate can be expressed using few NANDs, and so obtain the following claim (left as exercise): If (G, E, D) is a homomorphic encryption then there is an algorithm $EVAL$ that for every $(e, d) \leftarrow G(1^n)$, $x_1, \ldots, x_m \in \{0,1\}$, if $\hat{x}_i = E_e(x_i)$ and C is a Boolean circuit mapping $\{0,1\}^m$ to $\{0,1\}$, then

$$EVAL_e(C, \hat{x}_1, \ldots, \hat{x}_m) \approx_{|C|\mu(n)} E_e(C(x_1, \ldots, x_n))$$

where we say that $D \approx_\epsilon D'$ if their statistical distance is at most ϵ, μ is some negligible function, and $|C|$ denotes the number of gates of C.

In particular if C is polynomial size then these two distributions are statistically indistinguishable.

What is homomorphic encryption useful for? Canonical application is "cloud computing": Alice wants to store her file $x \in \{0,1\}^m$ on Bob's server. So she sends Bob $E_e(x_1) \cdots E_e(x_m)$. Then she wants to do computation on this file. For example, if the file is a database of people she may want to find out how many of them bought something in the last month. One way to do so would be for Alice to retrieve the entire file and do the computation on her own, but if she was able to handle this amount of communication and computation, perhaps she wouldn't have needed to use cloud computing in the first place.

Instead, Alice will ask Bob to perform this operation on the encrypted data, giving her an encryption of the answer, which she can of course decrypt. There is an issue of how Alice maintains integrity in this case, this is left as an exercise.

Note: this is a vast generalization of the cloud computing question in Homework 9, but in that question you are not allowed to use fully homomorphic encryption :(

Zero Knowledge from Homomorphic Encryption We've seen zero knowledge protocols for specific statements, but now we'll see such an encryption scheme for *any* statement, specifically for a public input circuit $C : \{0,1\}^m \to \{0,1\}$, we'll show a zero knowledge proof system (in fact even proof of knowledge) for the statement "there exists x such that $C(x) = 1$".

Note that this in some sense a tremendous overkill, since zero knowledge proofs for every statement can be based on just one-way functions, and the construction is not even terribly complicated, given basic NP-completeness results. But this protocol will give some intuition on homomorphic encryption, and will also be more communication efficient than the standard protocols. (Although there are PCP based protocols that are even more communication efficient.)

The basic protocol We'll describe the protocol in steps, starting with a simplified version that is not secure and tweaking it as we go along to ensure security.

Public Input: Boolean circuit $C : \{0,1\}^n \to \{0,1\}$.

Prover's private input: $x \in \{0,1\}^n$ such that $C(x) = 1$.

Step 1 Prover runs $(e, d) \leftarrow G(1^n)$, sends e to verifier.

Step 2 Prover sends $\hat{x} = E_e(x_1) \cdots E_e(x_n)$ to verifier.

Step 3 Verifier computes $\hat{c} = EVAL(C, \hat{x})$, sends $\hat{c}$ to prover.

Step 4 Prover sends $d = D_d(\hat{c})$ to verifier. Verifier accepts if $d = 1$.

Security This protocol is obviously not sound (can you see why?). We change it by having the verifier toss a coin $b \leftarrow_R \{0,1\}$ in Step 3. If $b = 1$ then the verifier proceeds as before. If $b = 0$ then the verifier sends $E_e(b)$ to the prover. The verifier checks in Step 4 that $b = d$.

Soundness We can now prove soundness of the new protocol though we will need a strengthening of the homomorphic encryption scheme, we require that it is possible to efficiently test that a public key e is in the range of the generation algorithm and a ciphertext $\hat{a}$ is in the range of the encryption algorithm. This can be fixed by adding another check by the verifier, though we'll defer details to the exercise.

Honest verifier zero knowledge This is not so hard to show. Note that the condition for honest verifier zero knowledge for a *private key* protocol is that one needs to generate the transcript *together with the corresponding randomness of the verifier*, since after all the verifier has of

course access to this randomness as well. But this can be done: the simulator will just send an encryption to junk and in Step 4 will send $d = b$.

Full zero knowledge It's not clear this protocol is zero knowledge also with respect to malicious verifiers. The problem is that such a verifier may use Step 3 to get an arbitrary query to a decryption box, and this is something he could not do on his own, thus violating at least the spirit of zero knowledge (and indeed one can find schemes in which one can show this is not zero knowledge). To fix this we slightly modify the protocol:

Step 4 The prover only sends a *commitment* to d (for example $f(x), r, \langle x, r\rangle \oplus d$, where f is a one-way permutation).

Step 5 Verifier sends all randomness it used in producing the ciphertext of Step 3. The prover verifies this is indeed the case, and otherwise aborts.

Step 6 The prover sends d and also the randomness used in producing the commitment.

This can be shown to preserve soundness, since soundness held even for *computationally unbounded* provers, and the commitment scheme is *perfectly binding*.

On the other hand, we can now have a simulator that will use rewinding, to extract the verifier's choice before answering the question.

Removing the "well formed" ciphertext and public key assumptions